НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»
ім. Ігоря Сікорського
Факультет інформатики та обчислювальної техніки
(повна назва інституту/факультету)

Кафедра обчислювальної техніки
(повна назва кафедри)

«До захисту допущено»
Завідувач кафедри
_____ _____
(підпис) (ініціали, прізвище)
“___”_____20__ р.

Дипломна робота

на здобуття ступеня бакалавра

з напряму підготовки 6.050102 Комп’ютерна інженерія
(код і назва)

на тему: Система аналізу та оптимізації транспортних пасажирських потоків__
_____
_____
_____

Виконав: студент __4__ курсу, групи ____ІО-53____
(шифр групи)

Задорожний Владислав Олегович _____ _____
(прізвище, ім’я, по батькові) (підпис)

Керівник професор, д.т.н., Новотарський М.А. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант нормоконтроль д.т.н., проф. Сімоненко В.П, _____
(назва розділу) (вчене звання та ступінь, прізвище, ініціали) (підпис)

Рецензент _____ _____ _____
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій дипломній роботі
немає запозичень з праць інших авторів
без відповідних посилань.
Студент _____
(підпис)

Київ – 2019 року

Національний технічний університет України
«Київський політехнічний інститут» ім. Ігоря Сікорського

Інститут (факультет) Факультет інформатики та обчислювальної техніки
(повна назва)

Кафедра Обчислювальної техніки
(повна назва)

Рівень вищої освіти – перший (бакалаврський)

Напрям підготовки 123 Комп'ютерна інженерія
(код і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ _____
(підпис) (ініціали, прізвище)
«___»_____20__ р.

ЗАВДАННЯ
на дипломну роботу студенту

Задорожному Владиславу Олеговичу _____
(прізвище, ім'я, по батькові)

1. Тема роботи Система аналізу та оптимізації транспортних пасажирських потоків _____ ,
керівник роботи професор, д.т.н., Новотарський М.А, _____ ,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від «____» _____ 201_ р. №_____

2. Термін подання студентом роботи _____ _____

3. Вихідні дані до роботи мова програмування Python, _____
_____
_____
_____

4. Зміст роботи

1. Огляд задачі та існуючих методів її вирішення _____
2. Вибір критеріїв оптимальності і алгоритму вирішення задачі _____
3. Опис роботи алгоритму _____
4. Порівняння з іншими алгоритмами _____

5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо)
1. Плакат «Завдання дипломної роботи» _____
2. Плакат «Ідея алгоритму колонії мурах для UTRP» _____
3. Плакат «Схема роботи алгоритму» _____

5. Дата видачі завдання _____

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1	Отримання завдання		
2	Збір інформації		
3	Огляд задачі та існуючих методів її вирішення		
4	Вибір критеріїв оптимальності та алгоритму вирішення задачі		
5	Опис роботи агоритму		
6	Порівняння з іншими алгоритмами		
7	Оформлення дипломної роботи		
8	Отримання допуску до захисту і подання роботи в ДЕК		

Студент _____ ___Задорожний В.О.
(підпис) (ініціали, прізвище)

Керівник роботи _____ ___Новотарський М.А.
(підпис) (ініціали, прізвище)

АНОТАЦІЯ

бакалаврської дипломної роботи Задорожного Владислава Олеговича

на тему «Система аналізу та оптимізації транспортних пасажирських потоків»

У даній роботі було поставлено завдання розробити алгоритм пошуку оптимального набору маршрутів громадського транспорту методами колективного інтелекту. Було проаналізовано існуючі критерії оптимізації та різні способи вирішення даної задачі.

Громадський транспорт відноситься до числа найважливіших галузей життєзабезпечення міста, від функціонування яких залежать якість життя населення, ефективність роботи галузей економіки міста та можливість використання її містобудівного та соціально-економічного потенціалу. Проте зараз в Україні громадський транспорт є малорозвиненим, а мережа маршрутів сформувалась у містах історично і вже не відповідає вимогам сучасності.

Головна мета – розробка алгоритму, що на основі матриці кореспонденцій і графа вулично-дорожньої мережі міг знайти оптимальний (або близький до оптимального) набір маршрутів громадського транспорту.

Результат роботи - реалізація алгоритму пошуку оптимального набору маршрутів громадського транспорту методами колективного інтелекту мовою Python та порівняння роботи цього алгоритму з іншими алгоритмами на тестовому прикладі.

Загальний обсяг роботи 62 сторінки, 5 рисунків, 9 таблиць, 27 посилань.

Ключові слова: громадський транспорт, колективний інтелект, алгоритм колонії мурах, ACA, задача пошуку набору оптимальних маршрутів громадського транспорту, UTRP

АНОТАЦИЯ

бакалаврской дипломной работы Задорожного Владислава Олеговича

на тему “Система анализа и оптимизации транспортных пассажирских потоков”

В данной работе была поставлена задача разработать алгоритм поиска оптимального набора маршрутов общественного транспорта методами коллективного интеллекта. Были проанализированы существующие критерии оптимизации и различные способы решения данной задачи.

Общественный транспорт относится к числу важнейших отраслей жизнеобеспечения города, от функционирования которых зависят качество жизни населения, эффективность работы отраслей экономики города и возможность использования ее градостроительного и социально-экономического потенциала. Однако сейчас в Украине общественный транспорт является малоразвитым, а сеть маршрутов сформировалась в городах исторически и уже не отвечает требованиям современности.

Главная цель – разработка алгоритма, на основе матрицы корреспонденций и графа улично-дорожной сети мог найти оптимальный (или близкий к оптимальному) набор маршрутов общественного транспорта.

Результат работы – реализация алгоритма поиска оптимального набора маршрутов общественного транспорта методами коллективного интеллекта языке Python и сравнения работы этого алгоритма с другими алгоритмами на тестовом примере.

Общий объем работы 62 страницы, 6 рисунков, 9 таблиц, 27 источников.

Ключевые слова: общественный транспорт, коллективный интеллект, алгоритм колонии муравьев, ACA, задача поиска набора оптимальных маршрутов общественного транспорта, UTRP

ABSTRACT

on Vladyslav Zadorozhnyi bachelor’s degree

thesis: “System for analysis and optimization of passenger traffic flows ”

The thesis concentrates on the problem of development of algorithm for solving the urban transit routing problem using collective intelligence methods. Different existing optimization criteria and approaches to solving of the problem is considered.

Public transport is one of the most important branches of city life support, from the operation of which considerably depend the quality of life, the efficiency of the industries of the city and the possibility of using its urban and socio-economic potential. But now in Ukraine public transport is insufficiently developed, the network of routes in cities formed mostly historically and no longer meets today’s requirements.

The main objective of the project is to develop an algorithm based on the correspondence matrix and graph the road network, that would be able to find an optimal (or close to optimal) set of public transport.

The developed algorithm for solving the urban transit routing problem is implemented with Python programming language and the results obtained by the algorithm are compared with results obtained by other algorithms on a test sample.

Bachelor's thesis size 62 pages, 6 pictures, 9 tables, 27 sources.
Keywords: public transport, collective intelligence, ants colony algorithm, ACA, urban transit routing problem, UTRP

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, ТЕРМІНІВ

ГТ – громадський транспорт

UTRP – (англ. urban transit routing problem) задача пошуку оптимального набору маршрутів громадського транспорту

UBND - (англ. urban bus network design) те саме, що і UTRP

UTNDP – (англ. urban transit network design problem) задача створення мережі громадського транспорту

ТПВ – транспортно-пересадочний вузол

TSP – (англ. travel salesman problem) задача комівояжера

ACA – (англ. ant colony algorithm) алгоритм колонії мурах

ВСТУП

Пасажирський транспорт відноситься до числа найважливіших галузей життєзабезпечення міста, від функціонування яких залежать якість життя населення, ефективність роботи галузей економіки міста та можливість використання її містобудівного та соціально-економічного потенціалу. За останні роки в багатьох містах України значно змінилася структура попиту на пасажирські перевезення. Це обумовлено процесом динамічного соціально-економічного розвитку міст, який викликав появу нових зон тяжіння пасажиропотоків, таких як ділові, торгові, розважальні та спортивні центри, зміни в структурі розселення жителів міста у зв'язку з появою нових зон активної житлової забудови.

Одночасно з ростом рівня автомобілізації населення зросло навантаження на дороги, на яких виникають «пробки». Все це зумовлює необхідність оптимізації стихійно сформованої системи громадського транспорту (ГТ), що є конкурентом автомобілям і є більш економічно-вигідною, але не відповідає потребам сьогоднішнього дня.

Ефективним вирішенням даної проблеми є застосування систем підтримки прийняття рішень в області побудови маршрутів громадського транспорту. Однак автоматизація завдань даної галузі вимагає проведення наукових досліджень з метою отримання ефективних алгоритмів, придатних для використання на практиці.

Ідея цієї роботи була мотивована насамперед проблемами міста Київ у сфері громадського транспорту.

Мета дипломної роботи – реалізація алгоритму пошуку оптимального набору маршрутів ГТ.

Завданням цієї дипломної роботи є:

1. Визначення критеріїв оптимальності набору маршрутів ГТ
2. Визначення методів пошуку оптимального набору маршрутів ГТ
3. Вибір методу пошуку оптимального набору маршрутів ГТ і його реалізація
4. Порівняння результатів роботи обраного методу з іншими методами на тестовому прикладі.

1. ОГЛЯД ЗАДАЧІ ТА ІСНУЮЧИХ МЕТОДІВ ЇЇ ВИРІШЕННЯ

У англомовній літературі задача знаходження оптимального набору маршрутів громадського транспорту називається здебільшого UTRP.

UTRP – це перша задача в рамках задачі створення мережі громадського транспорту (urban transit network design problem - UTNDP). Наступні задачі: вибір типів транспортних засобів для обслуговування маршрутів, створення розкладу руху та розкладів для працівників підприємств-перевізників [3].

Якщо аналізувати задачу пошуку оптимального набору маршрутів ГТ з боку обчислювальної складності, то цю задачу відносять до категорії NP-повних, що на великих обсягах даних вирішується здебільшого евристичними та метаевристичними методами – шляхом постійної генерації наборів маршрутів та їх валідації [1].

UTRP складно сформулювати математично, адже на відміну від задачі комівояжера (travelling salesman problem) маршрути можуть бути довгими чи короткими, прямими чи циклічними, одна зупинка може використовуватись різними маршрутами, зупинки можуть бути об’єднаними у ТПВ.

Крім генерації маршрутів також складної задачею є їх валідація, адже, наприклад, для знаходження оптимального шляху від початку до кінця подорожі по заданій маршрутній мережі та подальшого обрахунку значення критерію середнього часу подорожі пасажира потрібні доволі значні обчислювальні ресурси.

Але для вирішення задачі спочатку має бути визначено один чи кілька критеріїв оптимальності набору маршрутів ГТ. Лише після цього може бути

обраний метод, що буде використаний для формування оптимального набору маршрутів ГТ.

1.1 Принципи UTRP

Проектування мережі міських автобусів повинно ґрунтуватися на матриці кореспонденцій і бути направлено на полегшення поїздок і отримання прибутку для компаній-перевізників [6].

Принципами UTRP є [6]:

Ділянки з найбільшим попитом на поїздки повинні бути задоволені в першу чергу і мають отримати максимально пряме обслуговування.

- Коефіцієнт нелінійності руху (відношення довжини маршруту до найкоротшої відстані між початком і місцем призначення) повинен бути якомога меншим. Невеликий коефіцієнт нелінійності обслуговування призводить до зменшення відстані/часу поїздки.
- Мережа повинна бути добре доступною.
- Зона обслуговування має бути великою, щоб скоротити зони, що не обслуговуються, і забезпечити автобусне обслуговування в межах пішохідної доступності.
- Довжина маршруту повинна бути обмежена.
- Не повинно бути занадто довгих або занадто коротких маршрутів. Середнє число пасажирів на маршруті має бути більше мінімальної кількості.
- Першочергову увагу слід приділяти маршрутами з високим попитом.
- Поїздки по різних маршрутах повинні бути раціонально збалансованими.

1.2 Критерії оптимальності набору маршрутів ГТ

По суті, пасажирські перевезення громадським транспортом є частиною складної системи, яка складається не лише з пасажирів, а і з органів місцевого самоврядування, що будують інфраструктуру, а також компаній-перевізників, які є постачальниками послуг.

Але задача, яку ми розглядаємо, покриває лише проблему визначення оптимального набору маршрутів для пасажирів, а не їх економічну чи фінансову оптимальність в цілому. У цьому контексті критеріями оптимальності можуть бути:

- Відсоток поїздок без пересадок та з 1 (2, 3) пересадками
- Час подорожі (середній, медіанний тощо)

Тобто критерії, що напряму характеризують задоволення потреб пасажирів у перевезеннях.

Звісно, жоден з критеріїв не може використовуватися без урахування інтервалів руху транспортних засобів, що є наслідком економічної оптимальності. Проте для обрахунку значень критеріїв для вирішення цієї задачі в контексті інтервалів руху будуть використовуватись не показники, які будуть обраховані на наступних етапах вирішення задачі UTNDP, а спрощені теоретичні показники в залежності від пасажиропотоків маршрутів.

Жоден з критеріїв не може бути визначений як інтегральний, через що UTRP була визначена як задача багатокритеріальної оптимізації [4].

Тим більше на наступних етапах вирішення UTNDP різні набори маршрутів можуть мати різну оптимальність за іншими критеріями: економічними та фінансовими.

Проте критерій задоволення потреби пасажирів у перевезеннях є основним, тому ця задача і вирішується першою.

1.3 Методи пошуку оптимального набору маршрутів ГТ

Оскільки UTRP – це задача багатокритеріальної оптимізації, а сама задача визначена як NP-повна, то на великих обсягах даних вона вирішується здебільшого евристичними та метаевристичними методами – шляхом постійної генерації наборів маршрутів та їх валідації [4].

Такими методами є [4,5,7]:

- генетичні алгоритми (genetic algorithms)
- пошук з заборонами (tabu search)
- імітація випалювання (simulated annealing)
- алгоритми колективного інтелекту (swarm intelligence)

На більш ранніх етапах були популярні аналітичні методи на кшталт мінімальних покриваючих дерев на графах [4,5,7].

Але найбільш популярними останнім часом стали алгоритм колонії мурах та алгоритм колонії бджіл, які є представниками категорії алгоритмів колективного інтелекту (swarm intelligence або collective intelligence).

Популярність алгоритмів колективного інтелекту, можливо, пов'язана з тим, що на вибір члена колективу впливає не лише фактор індивідуальної оптимальності (наприклад, найкоротший шлях), а і фактор колективного вибору (кількість феромонів).

Вхідними даними для будь-якого з методів є дані про існуючу інфраструктуру (топологію): граф вулично-дорожньої мережі (ВДМ), час проїзду між вузлами графа ВДМ, зупинки громадського транспорту, попит на перевезення з однієї точки до іншої - матриця кореспонденцій [2, 3].

Граф ВДМ та час проїзду між вузлами графа для більшості міст можна узяти з сервісу OpenStreetMap, а матрицю кореспонденцій – з транспортної моделі певного міста.

Більш розширений огляд методів доступний за посиланнями [4,5,7].

1.4 Висновки до розділу 1

Наразі немає методу, який можна визнати однозначно найкращим для вирішення даної задачі, тому усі дослідження зводяться до експериментів з різними методами і критеріями оптимізації і порівняння їх ефективності на деяких тестових прикладах.

2. ВИБІР КРИТЕРІЇВ ОПТИМАЛЬНОСТІ І АЛГОРИТМУ ВИРІШЕННЯ ЗАДАЧІ

2.1 Вибір критеріїв оптимальності

Кожен з критеріїв оптимальності з розділу 1.2 має право бути застосованим, проте для цілей даної дипломної роботи було вирішено основним критерієм обрати критерій відсотку поїздок без пересадок. На відміну від середнього часу поїздки, він більш коректно відображає задоволеність попиту на поїздки, бо за досвідом, необхідність пересадок є негативним чинником, що зменшує задоволеність перевезеннями, навіть якщо це дозволяє зменшити час поїздки. До того ж, необхідність пересадок стимулює людину скористатись таким «безпересадковим» видом транспорту як особистий автомобіль, що є негативним явищем (насамперед для великих міст).

Другорядними (у наведеному далі порядку) критеріями при цьому будуть [8, 10, 11, 12]:

1. Відсоток поїздок з 1 пересадками
2. Відсоток поїздок з 2 пересадками
3. Відсоток незадоволених поїздок (в т.ч. з 3 і більше пересадок)
4. Середній час подорожі

2.2 Вибір алгоритму

Для вирішення задачі було вирішено обрати підхід на основі колективного інтелекту, а конкретно алгоритм колонії мурах з деякими покращеннями [6], що був застосований у місті Далянь, Китай.

Цей алгоритм був обраний з наступних міркувань.

У суті ГТ закладений принцип того, що ГТ не може бути транспортом «від дверей до дверей» (як, наприклад, індивідуальний транспорт чи таксі), але при цьому ГТ є економічно ефективним лише за умови певної кількості людей, що ним користується. При чому чим більше людей користується певним маршрутом ГТ, тим краще обслуговування людей – транспорт стає швидшим, інтервали зменшуються. В Україні це перехід від «маршруток» до автобусів великої та надвеликої місткості, далі – до тролейбусів, трамваїв, зчеплених трамваїв, що рухаються на виділений швидкісній інфраструктурі, а далі до метро, що наразі є найшвидшим і найбільш завантаженим видом ГТ [14].

Тобто принципи колективного інтелекту є подібними до принципів ГТ: оптимальний набір маршрутів ГТ намагається оптимізувати час поїздки (кількість пересадок тощо), але при цьому не може гарантувати для кожного пасажира індивідуально оптимальної поїздки, проте наявність прямого маршруту в місце призначення та більша швидкість руху досягається за рахунок більшої кількості користувачів, що колективно приймають рішення про користування тим чи іншим маршрутом ГТ [14].

2.3 Висновки до розділу 2

В цілому вибір критеріїв та методу вирішення задачі є доволі суб’єктивним через складність задачі, що пояснює різний вибір різних дослідників.

Водночас вибір критеріїв і алгоритму вирішення задачі був обраний автором роботи виходячи з власного досвіду і міркувань, а також специфічних особливостей інфраструктури міста Києва.

3. ОПИС РОБОТИ АЛГОРИТМУ

Обраний алгоритм оптимізації дозволяє максимізувати концентрацію прямих кореспонденцій на основі попиту в усій мережі.

Пряма кореспонденція – це поїздка без пересадок. А концентрація прямих кореспонденцій – це відношення кількості прямих кореспонденцій до довжини шляху.

Спочатку створюється порожня мережа, а потім додаються маршрути задля максимізації концентрації прямих кореспонденцій, вони додаються доти, доки всі пасажири не зможуть бути перевезеними або якщо деякі обмеження не будуть перевищені.

Метод не обмежується найкоротшими шляхами, але шукає шляхи з максимальною концентрацією прямих кореспонденцій з усіх можливих маршрутів. Це відбувається ще і тому, що пасажирів, що подорожують по найкоротшому шляху, не завжди найбільше.

Крім того, оскільки чим довший маршрут, тим більше пасажирів ним скористаються, тому деякі короткі шляхи можуть бути відкинуті, хоча вони і доволі популярні, якщо ми намагаємося максимізувати число прямих кореспонденцій.

Але при цьому, при прокладанні довших маршрутів загальна протяжність всієї мережі збільшується, що призводить до збільшення експлуатаційних витрат без повного використання мережі або рухомого складу.

Тому для підвищення ефективності роботи мережі в якості цільової функції використовується саме максимізація концентрація прямих кореспонденцій.

3.1 Модель оптимізації

Модель оптимізації набору маршрутів ГТ для обраного алгоритму виглядає наступним чином [6]:

$$\max D_{OD} = \frac{\sum_{i\in N}\sum_{j\in N} SP_{ij}x_{ij}}{\sum_{i\in N}\sum_{j\in N} \Delta_{ij} l_{ij} x_{ij}}$$

$$\begin{cases} L_{min} \le L \le L_{max} & (a) \\ Q^{sum} > Q_{min} & (b) \\ q^x \le q^x_{max} & (c) \\ b^n \le b^n_{max} & (d) \\ Q^{ij} < Q^{ij}_{max} & (e) \\ O, D \in F \end{cases} \quad (3.1)$$

де

(a) *Обмеження по довжині*: обмеження по довжині маршруту має бути узгоджено з трудовим законодавством конкретної країни і здоровим глуздом.

(b) *Обмеження щодо мінімальної кількості прямих кореспонденцій*: для забезпечення ефективності та вигоди компаніям-перевізникам повинно бути встановлено мінімальну кількість прямих кореспонденцій на прокладеному маршруті.

(c) *Обмеження нелінійності маршруту:* коефіцієнт нелінійності має бути меншим певного значення і розраховується так:

$$q^x_{OD} = L_{OD}/P^S_{OD}$$

де P^S_{OD}– довжина найкоротшого шляху між початком O і місцем призначення D.

(d) *Обмеження незбалансованості відрізка*: коефіцієнт незбалансованості відрізка є найбільшим відношенням максимального потоку відрізка з середнім потоку відрізка.

Він повинен бути меншим за певне значення і визначається як:

$$b^n = \frac{\max(Q^{ij} x_{ij})}{Q^{sum}/(COUNT-1)}$$

де *COUNT* — кількість зупинок на маршруті

(е) *Обмеження кількості пасажирів на ділянці:* кількість пасажирів на ділянці прокладеного маршруту повинна бути меншою за місткість ділянки, яка визначається місткістю автобуса і мінімальним інтервалом руху.

3.2 Алгоритм ACA

Алгоритм ACA, що лежить в основі обраного алгоритму, заснований на колонії і натхненний діями в пошуках їжі в колоніях мурашок [24, 25]. Алгоритм сфокусований не на математичному описі конкретних проблем, а скоріше на загальній можливості оптимізації і придатності для паралельних обчислень [6].

ACA є типовим прикладом застосування колективного інтелекту для вирішення проблем комбінаторної оптимізації. Його застосовували для вирішення TSP. Принципи алгоритму можна проілюструвати, вивчаючи процес пошуку їжі в колонії мурах. Тоді мурахи визначають свої пересування, оцінюючи концентрацію феромонів на шляху.

Цей процес можна описати як цикл з позитивним зворотним зв'язком, в якому чим більше мурах йдуть по сліду, тим більше феромонів там залишається, і тим вище ймовірність того, що по цьому сліду будуть слідувати інші мурахи [27]. Цей процес вибору є результатом самоактівізації мурах, в якому мурахи можуть знайти оптимальний шлях до кінцевих пунктів призначення в самому кінці.

Щоб зробити модель більш ефективною, пропонується нова стратегія для оновлення феромонів, яка враховує як глобальну, так і локальну інформацію, і розподіляє феромони по відрізках згідно концентрацій прямих кореспонденцій

на маршруті (глобальна інформація) і вкладу відрізка в маршрут (локальна інформація). Ця стратегія називається Ant-Weight [6].

Сам алгоритм може бути паралелізований природнім способом – усі пасажири розбиваються на колонії згідно з початком і кінцем їх подорожей, операції у різних колоніях можуть виконуватись паралельно, дані синхронізуються кожен крок або кожну ітерацію.

3.3 Послідовність роботи алгоритму

Якщо ми візьмемо автобуси як колонію мурах, початок як гніздо і кінець в якості джерела їжі, то завдання UBND може бути спрощена до процесу, за допомогою якого колонія мурах шукає їжу на основі феромонів, тобто це пошук оптимального маршруту автобусів від початку до місця призначення, ґрунтуючись на концентрації прямих кореспонденцій.

Нижче наведені конкретні кроки алгоритму. В порівнянні з оригінальним алгоритмом [6] реалізований алгоритм було дещо спрощено.

3.3.1 Ініціалізація

3.3.1.1 Ініціалізація автобусної мережі

Автобусна мережа складається з автобусних зупинок і відрізків шляху.

Відрізки шляху між зупинками є двонаправленими.

Початкова автобусна мережа представлена у вигляді графа.

Зупинки визначені з вхідних даних, а попит на перевезеннями між зупинками прив'язаний до відрізків графа.

3.3.1.2 Ініціалізація ACA

Колонія мурах ініціалізується шляхом створення m підколоній і визначенням їх топологічних структур, інтервалів часу (епох) і масштабів, після чого ініціалізується інтерфейс передачі повідомлень між підколоніями.

3.3.1.3 Ініціалізація матриці феромонів

Оскільки всі початкові зупинки однаково "приваблюють" автобуси, початкові ваги необхідно розмістити на всіх відрізках.

Оскільки концентрація прямих кореспонденцій розглядається як "феромон", ми використовуємо середню концентрацію прямих кореспонденцій для ініціалізації матриці феромонів.

Нарешті, на графі з'являються автобуси-першопрохідці.

Оскільки кожен шлях має однакову кількість феромонів, автобус може бути випадково поміщений на будь-який вузол, що розташований недалеко від гнізда (початкової точки).

3.3.2 Вибір можливих пар кінцевих зупинок

Перед пошуком шляху слід обрати початок O і місце призначення D. Ця пара вважається нежиттєздатною, якщо вона знаходиться на маршруті, який вже був прокладений або не задовольняє обмеженням. Після цього кожна підколонія починає пошук маршрутів між парами OD незалежно.

3.3.3 Пошук маршруту

Наступна зупинка l буде обрана з безлічі зупинок за правилом переходу. Ухвалення рішення про наступній зупинці ґрунтується на імовірнісному правилі (правило переходів) з урахуванням як концентрації феромонів , так і значення видимості відповідного ребра.

В даному випадкує похідною від оновлення феромонів на кожному ребрі після кожного циклу, ґрунтуючись на правилі переходу, про яке говорилося раніше; в той час як є похідною від «жадібного» методу, який заохочує автобус до відвідування локально оптимального шляху.

Значення фіксоване в кожному циклі, крім того, в задачі UBND відрізки між зупинками є асиметричними, тобто не завжди дорівнює . Тому ми використовуємо напрямлений "феромон", тобто НЕ дорівнює .

Щоб зробити пошук ефективним, ми вважаємо, що мурахи відчувають напрямок і здатні відчути позицію їжі.

Тоді колонія мурах починає використовувати нове локальне правило пошуку, яке вважає, що мурахи схильні вибирати більш короткий шлях, тобто вони вибирають тільки зупинку з меншою відстанню до продовольства в порівнянні з поточною зупинкою.

Ми визначаємо ймовірність того, що автобус переміститься від зупинки і до зупинки j за формулою (3.5):

$$p_{ij}(k) = \begin{cases} \frac{\tau_{ij}^{\alpha} \times \eta_{ij}^{\beta}}{\sum_{h \in tabu_k} \tau_{ih}^{\alpha} \times \eta_{ih}^{\beta}}, \text{якщо } j \notin tabu_k \\ 0, \qquad \text{інакше} \end{cases} \quad (3.5)$$

де:

$p_{ij}(k)$ – ймовірність вибору j-тої зупинки k-м автобусом;

τ_{ij} і η_{ij}- концентрація феромонів і видимість ребра (i, j);

α і β - відносний вплив феромонних слідів і значень видимості;

$tabu_k$ - набір недосяжних (тобто вже відвіданих) вузлів для k-того автобуса.

Кількість пасажирів на відрізку означає кількість пасажирів, які проїжджають через цей відрізок.

Обчислення загального числа пасажирів на маршруті за формулою (3.6):

$$Q^{sum} = \sum_{(k,l)\in S_{OD}} Q^{kl} \quad (3.6)$$

Обчислення довжини маршруту за формулою (3.7):

$$L = \sum_{i\in N} \sum_{j\in N} \Delta_{ij} l_{ij} x_{ij} \quad (3.7)$$

Розрахунок концентрації прямих кореспонденцій на маршруті за формулою (3.8):

$$D_{OD} = Q^{sum}/L \quad (3.8)$$

Кожен маршрут після розрахунку оцінюється.

Якщо маршрути задовольняють обмеженням довжини, мінімальної кількості прямих кореспонденцій і обмеження по коефіцієнту нелінійності, то мережа автобусів буде відповідним чином оновлена; в іншому випадку маршрут буде відкинутий, і алгоритм повернеться до пункту 3.3.2.

3.3.4 Правило оновлення феромонів

Після прокладання маршруту необхідно оновити феромони в мережі.

При пошуку маршруту оптимізація мережі в повній мірі залежить від феромону і видимості . Видимість є порівняно стабільною, тому феромони набувають особливо важливе значення з точки зору пошуку нових маршрутів.

Найбільш поширеною стратегією для розрахунку приросту феромонів є метод Ant-Circle [6], який включає в себе глобальну інформацію. Однак, оскільки він ігнорує локальну інформацію, швидкість збіжності і якість оптимізації не є хорошими.

Зокрема, стратегія відновлення для обчислення приросту феромонів реалізується за формулою (3.9):

$$\Delta\tau_{ij}^{k} = \begin{cases} \frac{Q}{f^k} \times \frac{f^k - f_{ij}}{(n-2)f^k}, & \text{якщо відрізок } (i,j) \text{ є у маршруті } k \\ 0, & \text{інакше} \end{cases} \quad (3.9)$$

де

$\Delta\tau_{ij}^{k}$ - приріст феромонів на відрізку (i, j) маршруту k, виявленого мурахою

Q – константа

f^k- концентрація прямих кореспонденцій маршруту k

f_{ij} - концентрація прямих кореспонденцій на відрізку (i, j)

n = число зупинок уздовж маршруту *k (n > 2)*.

Стратегія поновлення феромонів складається з двох компонентів: перший глобальне збільшення феромона пов'язане з концентрацією прямих кореспонденцій на маршруті k; другий - локальне збільшення феромонів, засноване на вкладі відрізка (i, j) в маршрут k. Таким чином, стратегія відновлення гарантує, що присвоєний приріст феромонів безпосередньо пропорційний концентрації прямих кореспонденцій.

Тим часом, шляхом коригування методу присвоєння феромонів для відрізків поточного оптимального шляху, алгоритм може посприяти більш точному пошуку в наступному циклі в більш сприятливому районі. Це допомагає поліпшити здатність до навчання алгоритму з досвіду минулих пошуків і підвищити ефективність.

Крім того, щоб уникнути локальної оптимізації і для збільшення ймовірності отримання більш високоякісних рішень, верхні і нижні межі $[\tau_{min}, \tau_{max}]$ фіксуються в рівнянні поновлення [26].

Тому феромони відрізка після циклу мають бути оновлені за формулами (3.10-3.12):

$$\tau_{ij}^{new} = \rho \times \tau_{ij}^{old} + \sum_{k=1}^{m} \Delta\tau_{ij}^{k}, \rho \in (0,1) \quad (3.10)$$

$$\tau_{min} = \min(f_{ij}), \tau_{max} = \max(f_{ij}) \quad (3.11)$$

$$\tau_{ij} = \begin{cases} \max(\tau_{min}, \tau_{ij}^{new}), \tau_{ij}^{new} < \bar{\tau} \\ \min(\tau_{max}, \tau_{ij}^{new}), \tau_{ij}^{new} \geq \bar{\tau} \end{cases} \quad (3.12)$$

де

τ_{ij}^{new}- феромони на відрізку (i, j) після поновлення;

τ_{ij}^{old}- феромони на відрізку (i, j) перед оновленням

ρ - константа, яка контролює швидкість випаровування

3.3.5 Генерація обраних маршрутів

Після того як всі мурахи завершили цикл, маршрут з найбільшою концентрацією прямих кореспонденцій між OD вибирається в якості обраного маршруту. Зупинки, які він покриває, послідовно додаються в S_{OD}.

Таким чином, пошук маршрутів між OD завершується, і алгоритм повернеться до Кроку 2 для наступного пошуку маршруту між OD. Цей процес повторюється до тих пір, поки не будуть знайдені всі допустимі маршрути між парами OD.

Таким чином, між допустимими OD створюється не більше одного обраного маршруту.

Множина маршрутів позначається S.

3.3.6 Прокладання маршрутів

Маршрут з найбільшою концентрацією в множині S потім вибирається для додавання в мережу, після чого інші відкидаються, а відповідні дані оновлюються.

3.3.7 Перевірка мережі

Після прокладання маршруту кількість подорожуючих, які перевозяться по маршруту, має бути вирахувано з вихідної матриці кореспонденцій.

По-перше, обчислюється загальна кількість пасажирів по кожній ділянці маршруту і його ємність. Якщо пропускна здатність кожного відрізка маршруту переважує потік, то повинні перевозитися всі пасажири на маршруті, а число пасажирів між зупинками має бути вирахувано з матриці кореспонденцій. Якщо пропускна здатність відрізка менше, ніж потік, з матриці віднімається тільки кількість перевезених пасажирів. Для спрощення будемо вважати пропускну здатність усіх відрізків необмеженою.

Наступним кроком є визначення того, чи слід завершувати роботу чи ні.

3.3.8 Умова завершення роботи

Якщо немає маршруту, що задовольняє обмеженням або кількість ітерацій досягає максимуму, то закінчуємо; в іншому випадку поверніться до кроку 2.

3.4 Схема роботи алгоритму

На рисунку 3.1 показана вся схема алгоритму.

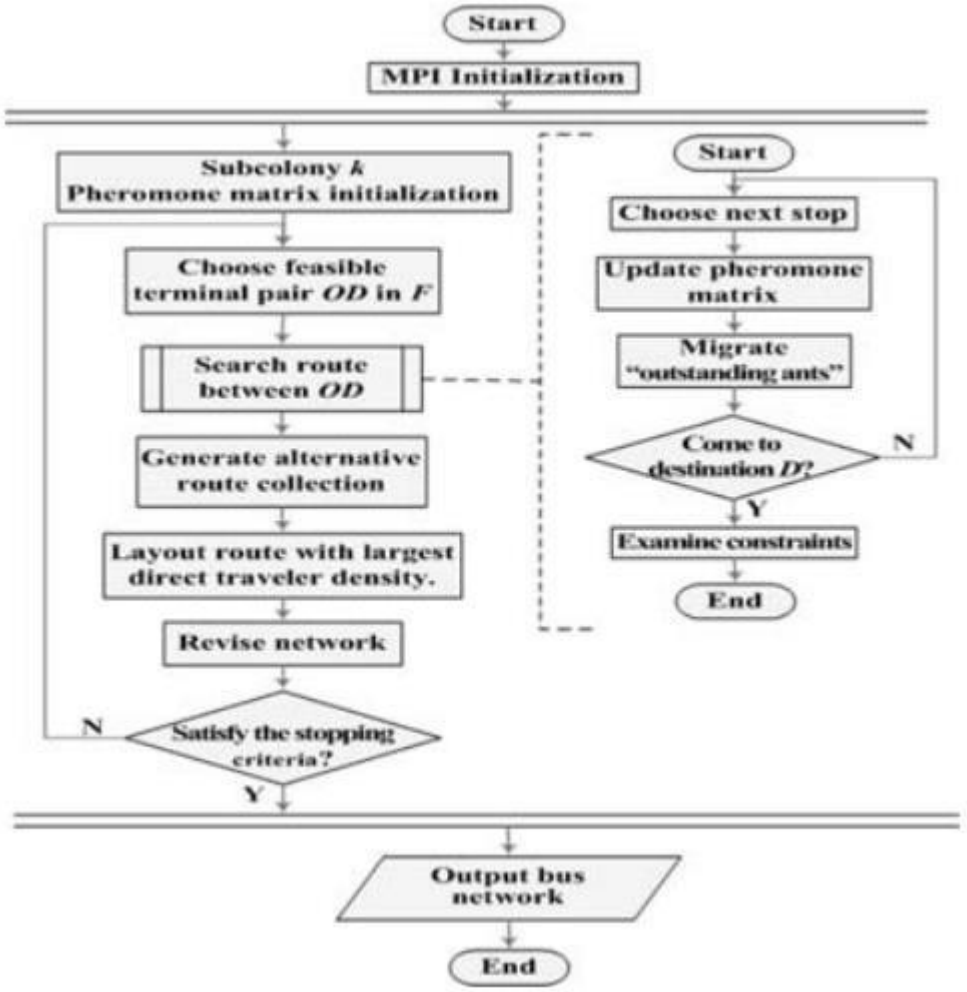

Рисунок 3.1. Схема роботи алгоритму [6]

3.5 Висновки до розділу 3

Для реалізації був обраний алгоритм ACA [6], який представляє мурах як автобуси, що рухаються від гнізда до їжі, з використанням покращенного алгоритму оновлення феромонів Ant-Weight. Схема роботи алгоритму зображена на рисунку 3.1, а сам відбір маршрутів має задовольняти обмеження з формули 3.1.

4. ПОРІВНЯННЯ З ІНШИМИ АЛГОРИТМАМИ

Реалізований алгоритм був протестований і порівняний з іншими алгоритмами на 1 тестовому прикладі Mandl’s Swiss Network (15 вузлів і 20 відрізків) з [8, 10-12].

Нагадаю, що критеріями оптимальності є [8, 10, 11, 12]:

1. Відсоток поїздок без пересадок
2. Відсоток поїздок з 1 пересадками
3. Відсоток поїздок з 2 пересадками
4. Відсоток незадоволених поїздок (в т.ч. з 3 і більше пересадок)
5. Середній час подорожі

Параметри алгоритму ACA були обрані емпіричним шляхом. З усіх обмежень для маршрутів застосовується лише те, що коефіцієнт нелінійності всього маршруту має бути не більшим за 1,5, також не використовувались межі для обсягу феромонів і були відсутні «мігруючі» мурахи.

4.1 Результати роботи алгоритму і порівняння

Таблиця 4.1 – Порівняння роботи алгоритмів за вказаними критеріями при виборі 4 маршрутів

Алгоритм	(%)	(%)	(%)	(%)	( )
Mandl [16]	69,94	29,93	0,13	0,00	12,90
Kidwai [18]	72,95	26,91	0,13	0,00	12,72
Chakroborty and Wivedi [19]	86,86	12,00	1,14	0,00	11,90
Fan and Mumford, best [13]	93,26	6,74	0,00	0,00	11,37
Fan and Mumford, HC, average [13]	91,83	8,17	0,00	0,00	11,69
Fan and Mumford, SA, average [13]	92,48	7,52	0,00	0,00	11,55
Fan, Mumford and Evans [10]	90,88	8,35	0,77	0,00	10,65
Zhang, Lu and Fan [20]	91,46	8,54	0,00	0,00	10,65
Chew and Lee, best [21]	93,71	6,29	0,00	0,00	10,82
Chew and Lee, average [21]	92,88	6,91	0,20	0,00	11,16
PSO best [11]	91,84	7,64	0,51	0,00	10,64
PSO average [11]	90,52	8,75	0,73	0,00	10,71
PSO median [11]	90,56	8,67	0,83	0,00	10,67
PSO worst [11]	90,56	8,61	0,83	0,00	10,85
Дана робота	**90,30**	**8,22**	**1,48**	**0,00**	**12,10**

Таблиця 4.2 – Порівняння роботи алгоритмів за вказаними критеріями при виборі 6 маршрутів

Алгоритм	(%)	(%)	(%)	(%)	( )
Baaj and Mahmassani [17]	78,61	21,39	0,00	0,00	11,86
Kidwai [18]	77,92	19,62	2,40	0,00	11,87
Chakroborty and Wivedi [19]	86,04	13,96	0,00	0,00	10,30
Fan and Mumford, best [13]	91,52	8,48	0,00	0,00	10,48
Fan and Mumford, HCb, average [13]	90,23	9,26	0,51	0,00	10,78
Fan and Mumford, SAc, average [13]	90,87	8,74	0,39	0,00	10,65
Fan, Mumford and Evans [10]	93,19	6,23	0,58	0,00	10,46
Zhang, Lu and Fan [20]	91,12	8,88	0,00	0,00	10,50
Chew and Lee, best [21]	95,57	4,43	0,00	0,00	10,28
Chew and Lee, average [21]	93,85	5,88	0,24	0,03	10,51
PSO best [11]	96,21	3,47	0,32	0,00	10,23
PSO average [11]	95,62	4,28	0,10	0,00	10,28
PSO median [11]	95,79	4,11	0,00	0,00	10,27
PSO worst [11]	94,09	5,27	0,64	0,00	10,37
Дана робота	**93,32**	**5,52**	**1,12**	**0,00**	**11,85**

Таблиця 4.3 – Порівняння роботи алгоритмів за вказаними критеріями при виборі 7 маршрутів

Алгоритм	(%)	(%)	(%)	(%)	( )
Baaj and Mahmassani [10]	80,99	19,01	0,00	0,00	12,5
Kidwai [34]	93,91	6,09	0,00	0,00	10,70
Chakroborty and Wivedi [15]	89,15	10,85	0,00	0,00	10,15
Fan and Mumford, best [13]	93,32	6,36	0,32	0,00	10,42
Fan and Mumford, HCb, average [13]	92,21	7,13	0,66	0,00	10,74
Fan and Mumford, SAc, average [13]	92,47	6,95	0,58	0,00	10,62
Fan, Mumford and Evans [10]	92,55	6,68	0,77	0,00	10,44
Zhang, Lu and Fan [20]	92,89	7,11	0,00	0,00	10,46
Chew and Lee, best [21]	95,57	4,43	0,00	0,00	10,27
Chew and Lee, average [21]	96,47	3,53	0,00	0,00	10,31
PSO best [11]	97,17	2,83	0,00	0,00	10,16
PSO average [11]	96,55	3,45	0,01	0,00	10,23
PSO median [11]	96,60	3,40	0,00	0,00	10,20
PSO worst [11]	95,63	4,37	0,00	0,00	10,40
Дана робота	**90,30**	**8,22**	**1,48**	**0,00**	**11,56**

4.2 Висновки до розділу 4

Як видно з таблиць, реалізований алгоритм в середньому подібний до більшості сучасних алгоритмів, а при генеруванні 6 маршрутів дає результат близький до найкращого.

Проте очевидно, що алгоритм програє іншим у середньому часі на поїздку, а при генерації 7 маршрутів раптово втрачає у оптимальності у показнику кількості поїздок без пересадок. І те, і інше можна пояснити тим, що шлях з пересадкою для деяких пасажирів є менш довгим за рахунок великого коефіцієнту нелінійності маршрутів, а для решти є достатньо довгим (порівняно з найкоротшим шляхом взагалі) і шлях без пересадок.

Тож саме зменшення граничного порогу коефіцієнту нелінійності маршрутів (і обмеження коефіцієнта нелінійності окремих частин маршруту) може у подальшому покращити результати.

Також покращити результати може введення «штучних» пасажирів між парами початок-кінець між якими ніхто не подорожує. Це пов’язано з тим, що в поточній реалізації маршрут між такими парами кінцевих не може існувати, хоча теоретично це міг бути доволі ефективний маршрут для проміжних пасажирів.

5. ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ПРОГРАМНОГО ПРОДУКТУ

5.1 Вступ

У даному розділі проводиться оцінка основних характеристик програмного продукту, призначеного для аналізу нелінійних нестаціонарних процесів. Інтерфейс користувача був розроблений за допомогою мови програмування Python у середовищі розробки JetBrains PyCharm.

Програмний продукт призначено для використання на персональних комп’ютерах під управлінням операційної системи Windows, Linux або MacOS.

Нижче наведено аналіз різних варіантів реалізації модулю з метою вибору оптимальної, з огляду при цьому як на економічні фактори, так і на характеристики продукту, що впливають на продуктивність роботи і на його сумісність з апаратним забезпеченням. Для цього було використано апарат функціонально-вартісного аналізу.

Функціонально-вартісний аналіз (ФВА) – це технологія, яка дозволяє оцінити реальну вартість продукту або послуги незалежно від організаційної структури компанії. Як прямі, так і побічні витрати розподіляються по продуктам та послугам у залежності від потрібних на кожному етапі виробництва обсягів ресурсів. Виконані на цих етапах дії у контексті метода ФВА називаються функціями.

Мета ФВА полягає у забезпеченні правильного розподілу ресурсів, виділених на виробництво продукції або надання послуг, на прямі та непрямі витрати. У даному випадку – аналізу функцій програмного продукту й виявлення усіх витрат на реалізацію цих функцій.

Фактично цей метод працює за таким алгоритмом:

визначається послідовність функцій, необхідних для виробництва продукту. Спочатку – всі можливі, потім вони розподіляються по двом групам: ті, що впливають на вартість продукту і ті, що не впливають. На цьому ж етапі оптимізується сама послідовність скороченням кроків, що не впливають на цінність і відповідно витрат.

для кожної функції визначаються повні річні витрати й кількість робочих часів.

для кожної функції наоснові оцінок попереднього пункту визначається кількісна характеристика джерел витрат.

після того, як для кожної функції будуть визначені їх джерела витрат, проводиться кінцевий розрахунок витрат на виробництво продукту.

5.2 Постановка задачі техніко-економічного аналізу

У роботі застосовується метод ФВА для проведення техніко-економічний аналізу розробки системи аналізу нелінійних нестаціонарних процесів.
Оскільки основні проектні рішення стосуються всієї системи, кожна окрема підсистема має їм задовольняти. Тому фактичний аналіз представляє собою аналіз функцій програмного продукту, призначеного для збору, обробки та проведення аналізу гетероскедастичних процесів в економіці та фінансах.

Відповідно цьому варто обирати і систему показників якості програмного продукту.

Технічні вимоги до продукту наступні:

програмний продукт повинен функціонувати на персональних комп’ютерах із стандартним набором компонент;

забезпечувати високу швидкість обробки великих об’ємів даних у реальному часі;

забезпечувати зручність і простоту взаємодії з користувачем або з розробником програмного забезпечення у випадку використовування його як модуля;

передбачати мінімальні витрати на впровадження програмного продукту.

5.2.1 Обґрунтування функцій програмного продукту

Головна функція 0 – розробка програмного продукту, який аналізує процес за вхідними даними та будує його модель для подальшого прогнозування. Виходячи з конкретної мети, можна виділити наступні основні функції ПП:

1 – вибір мови програмування;
2 – інтерфейс користувача.
3 – введення вхідних даних;

Кожна з основних функцій може мати декілька варіантів реалізації.

Функція F_1:

а) мова програмування Python;

б) мова програмування C++;

Функція F_2:

а) текстовий інтерфейс користувача;

б) графічний інтерфейс користувача.

Функція F_3:

а) введення даних вручну (з файлу з певним форматом даних);

б) генерація вхідних даних.

5.2.2 Варіанти реалізації основних функцій

Варіанти реалізації основних функцій наведені у морфологічній карті системи (рис. 4.1). На основі цієї карти побудовано позитивно-негативну матрицю варіантів основних функцій (таблиця 4.1).

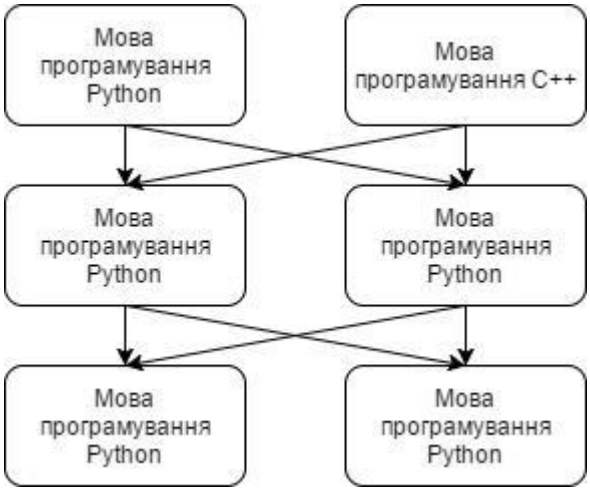

Рисунок 5.1 – Морфологічна карта

Морфологічна карта відображує всі можливі комбінації варіантів реалізації функцій, які складають повну множину варіантів ПП.

Таблиця 5.1 – Позитивно-негативна матриця

Основні функції	Варіанти реалізації	Переваги	Недоліки
F1	*А*	Займає менше часу при написанні коду, кросплатформений	Повільно виконується
	Б	Код швидко виконується, не кросплатформений	Займає більше часу при написанні коду
F2	*А*	Легкий у створенні	-
	Б	-	Складний у створенні
F3	*А*	Можливість порівняння роботи з іншими алгоритмами	Обмеженість у тестуванні
	Б	Широкі можливості для тестування	Складність реалізації, відсутність даних для порівняння роботи з іншими алгоритмами

На основі аналізу позитивно-негативної матриці робимо висновок, що при розробці програмного продукту деякі варіанти реалізації функцій варто відкинути, тому, що вони не відповідають поставленим перед програмним продуктом задачам. Ці варіанти відзначені у морфологічній карті.

Функція *F1:*

Оскільки на першому етапі критичною є можливість швидкого написання коду, то варіант б) має бути відкинутий, проте у подальшому розрахунки можуть проводитись з великими об’ємами вхідних даних і час виконання

програмного коду є дуже необхідним, тому потім має бути використаний саме варіант б).

Функція *F2:*

Інтерфейс користувача не відіграє велику роль у даному програмному продукту, тому вважаємо варіант а) більш доцільним.

Функція *F3:*

Оскільки важливої частиною задачі є порівняння роботи алгоритму з іншими алгоритмами, то є сенс використати варіант а), проте для тестування алгоритму є важливим також і варіант б).

Таким чином, будемо розглядати такі варіанти реалізації ПП:

1. F1а – F2а – F3а

2. F1а – F2а – F3б

Для оцінювання якості розглянутих функцій обрана система параметрів, описана нижче.

5.3 Обґрунтування системи параметрів ПП

5.3.1 Опис параметрів

На підставі даних про основні функції, що повинен реалізувати програмний продукт, вимог до нього, визначаються основні параметри виробу, що будуть використані для розрахунку коефіцієнта технічного рівня.

Для того, щоб охарактеризувати програмний продукт, будемо використовувати наступні параметри:

X1 – швидкодія мови програмування;

X2 – об’єм пам’яті для збереження даних;

X3 – час обробки даних;

X4 – потенційний об’єм програмного коду.

X1: Відображає швидкодію операцій залежно від обраної мови програмування.

X2: Відображає об’єм пам’яті в оперативній пам’яті персонального комп’ютера, необхідний для збереження та обробки даних під час виконання програми.

X3: Відображає час, який витрачається на дії.

X4: Показує розмір програмного коду який необхідно створити безпосередньо розробнику.

5.3.2 Кількісна оцінка параметрів

Гірші, середні і кращі значення параметрів вибираються на основі вимог замовника й умов, що характеризують експлуатацію ПП як показано у табл. 5.2.

Таблиця 5.2 – Основні параметри ПП

Назва параметра	**Умовні позначення**	**Одиниці виміру**	**Значення параметра**		
			гірші	**середні**	**кращі**
Швидкодія мови програмування	X1	Оп/мс	20000	10000	2000
Об’єм пам’яті для збереження даних	X2	Мб	32	16	8
Час обробки даних алгоритмом	X3	мс	1000	500	100
Потенційний об’єм програмного коду	X4	кількість строк коду	1000	750	150

За даними таблиці 5.2 будуються графічні характеристики параметрів – рис. 5.2 – рис. 5.5.

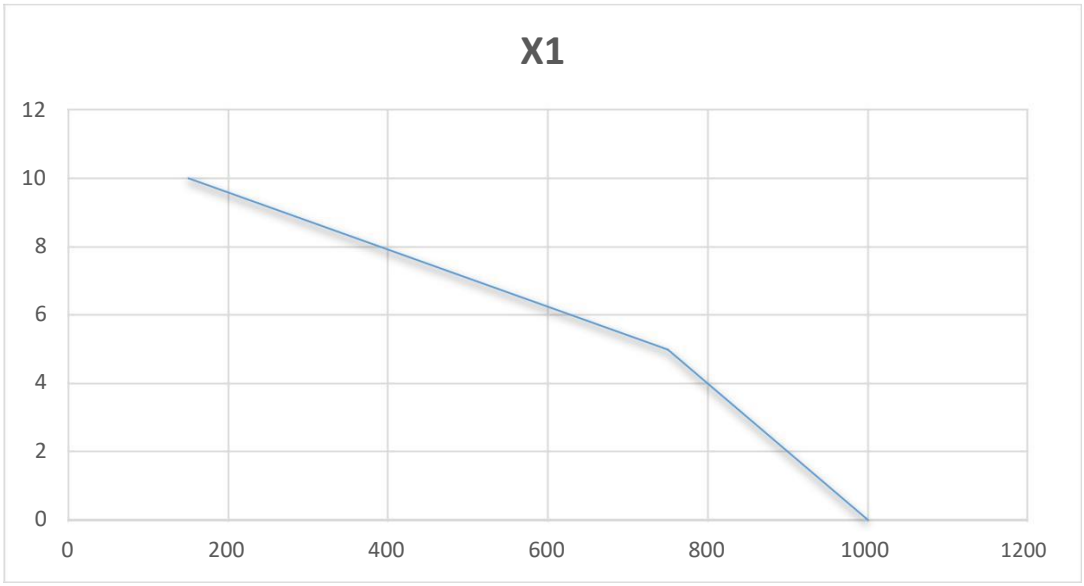

Рисунок 5.2 – X1, швидкодія мови програмування

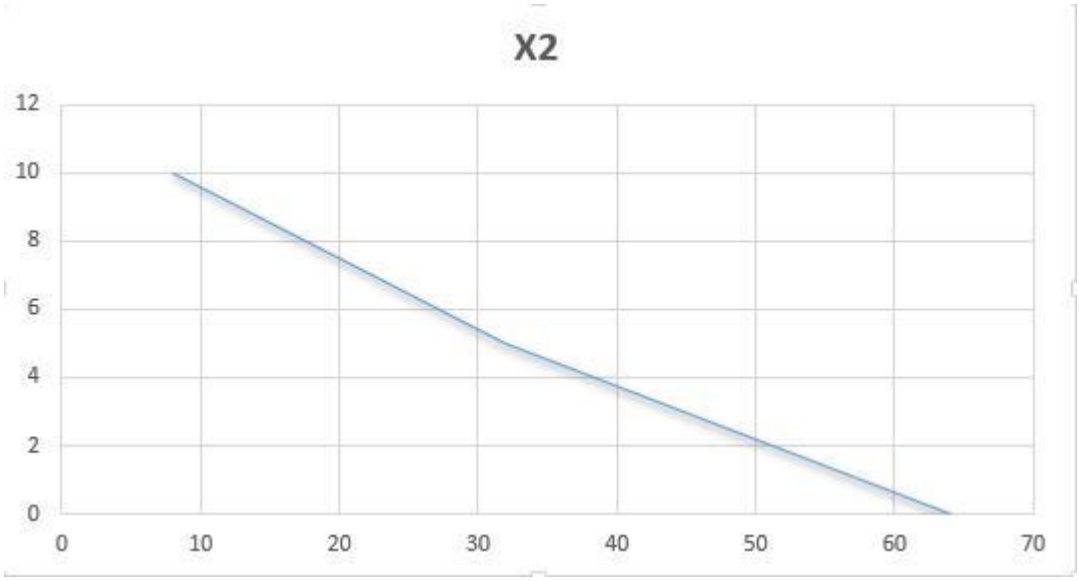

Рисунок 5.3 – X2, об’єм пам’яті для збереження даних

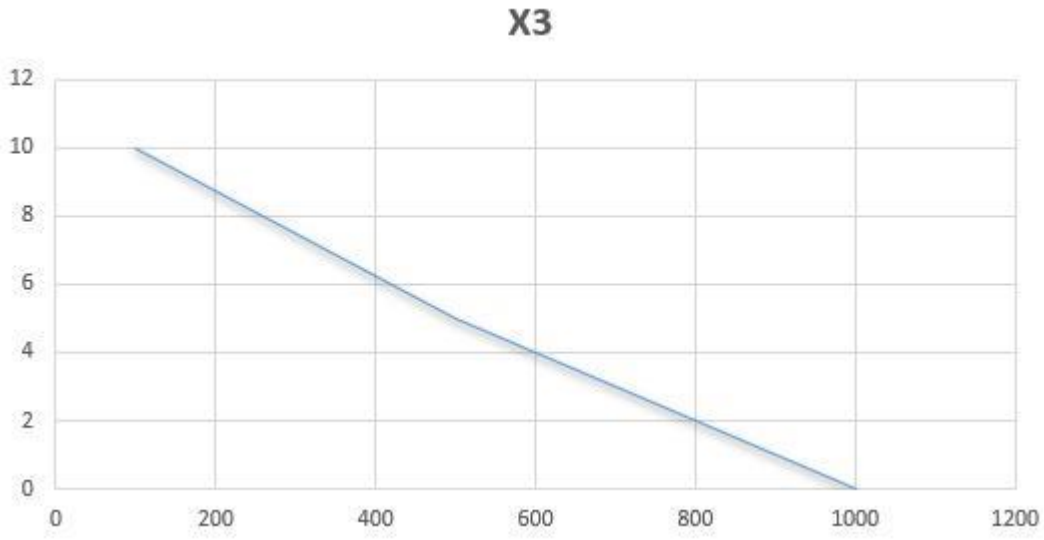

Рисунок 5.4 – X3, час обробки даних алгоритмом

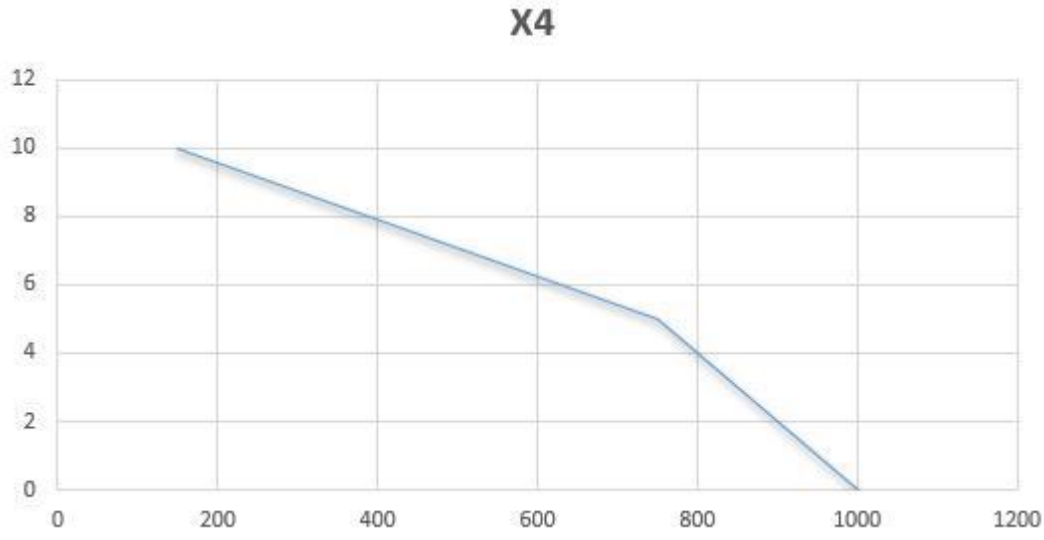

Рисунок 5.5 – X4, потенційний об’єм програмного коду

5.3.3 Аналіз експертного оцінювання параметрів

Після детального обговорення й аналізу кожний експерт оцінює ступінь важливості кожного параметру для конкретно поставленої цілі – розробка програмного продукту, який дає найбільш точні результати при знаходженні параметрів моделей адаптивного прогнозування і обчислення прогнозних значень.

Значимість кожного параметра визначається методом попарного порівняння. Оцінку проводить експертна комісія із 7 людей. Визначення коефіцієнтів значимості передбачає:

визначення рівня значимості параметра шляхом присвоєння різних рангів;

перевірку придатності експертних оцінок для подальшого використання;

визначення оцінки попарного пріоритету параметрів;

обробку результатів та визначення коефіцієнту значимості.

Результати експертного ранжування наведені у таблиці 5.3.

Таблиця 5.3 – Результати ранжування параметрів

Позначення параметра	**Назва параметра**	**Одиниці виміру**	**Ранг параметра за оцінкою експерта**							**Сума рангів** R_i	**Відхилення** i	i^2
			1	2	3	4	5	6	7			
X1	Швидкодія мови програмування	Оп/мс	4	3	4	4	4	4	4	27	0,75	0,56
X2	Об’єм пам’яті для збереження даних	Мб	4	4	4	3	4	3	3	25	-1,25	1,56
X3	Час обробки даних алгоритмом	Мс	2	2	1	2	1	2	2	12	-14,25	203,06
X4	Потенційний об’єм програмного коду	кількість строк коду	5	6	6	6	6	6	6	41	14,75	217,56
	Разом		15	15	15	15	15	15	15	105	0	420,75

Для перевірки степені достовірності експертних оцінок, визначимо наступні параметри:

а) сума рангів кожного з параметрів і загальна сума рангів за формулою (5.1):

$$= \sum = \frac{(+1)}{} = 105, \quad (5.1)$$

де N – число експертів од;

n – кількість параметрів, од;

– ранг, од.

б) середня сума рангів за формулою (5.2):

$$= \frac{1}{} = 26{,}25, \quad (5.2)$$

де n – кількість параметрів, од;

– ранг, од.

в) відхилення суми рангів кожного параметра від середньої суми рангів за формулою (5.3):

$$\Delta = - , \quad (5.3)$$

де Т – середня сума рангів, од;
– сума рангів кожного параметра, од.

Сума відхилень по всім параметрам повинна дорівнювати

0; г) загальна сума квадратів відхилення за формулою (5.4):

$$= \sum \Delta^2 = 420{,}75, \quad (5.4)$$

де Δ^2 – відхилення, од;

Порахуємо коефіцієнт узгодженості за формулою (5.5):

$$= \frac{12}{^{2}(^{3}-)} = \frac{12 \cdot 420{,}75}{7^{2}(5^{3} \quad -5)} = 1{,}03 > = 0{,}67, \quad (5.5)$$

де S – загальна сума квадратів відхилення, од;

– нормативний коефіцієнт узгодженості, од.

Ранжування можна вважати достовірним, тому що знайдений коефіцієнт узгодженості перевищує нормативний, котрий дорівнює 0,67.

Скориставшись результатами ранжирування, проведемо попарне порівняння всіх параметрів і результати занесемо у таблицю 5.4.

Таблиця 5.4 – Попарне порівняння параметрів

Параметри	Експерти							Кінцева оцінка	Числове значення
	1	2	3	4	5	6	7		
X1 і X2	=	>	=	<	=	<	<	<	0,5
X1 і X3	<	<	<	<	<	<	<	<	0,5
X1 і X4	>	>	>	>	>	>	>	>	1,5
X2 і X3	<	<	<	<	<	<	<	<	0,5
X2 і X4	>	>	>	>	>	>	>	>	1,5
X3 і X4	>	>	>	>	>	>	>	>	1,5

Числове значення, що визначає ступінь переваги *i*–го параметра над *j*–тим, a_{ij} визначається по формулі (5.6):

$$a_{ij} \begin{cases} 1{,}5 \text{ при } X_i > X_j \\ 1.0 \text{ при } X_i = X_j \\ 0.5 \text{ при } X_i < X_j \end{cases} \qquad (5.6)$$

З отриманих числових оцінок переваги складемо матрицю $A = \| a_{ij} \|$.

Для кожного параметра зробимо розрахунок вагомості $K_{вi}$ за наступною формулою (5.7):

$$= \frac{}{}, \text{де } = \Sigma \quad , \qquad (5.7)$$

Відносні оцінки розраховуються декілька разів доти, поки наступні значення не будуть незначно відрізнятися від попередніх (менше 2%).На другому і наступних кроках відносні оцінки розраховуються за наступними формулами (5.8, 5.9):

$$= \frac{}{} \quad , \qquad (5.8)$$

$$\text{де } ' = \Sigma \quad , \qquad (5.9)$$

Як видно з таблиці 5.5, різниця значень коефіцієнтів вагомості не перевищує 2%, тому більшої кількості ітерацій не потрібно.

Таблиця 5.5 – Розрахунок вагомості параметрів

Параметри	Параметри				Перша ітер.		Друга ітер.		Третя ітер	
	X1	X2	X3	X4						
X1	1,0	0,5	0,5	1,5	3,5	0,219	22,25	0,216	100	0,215
X2	1,5	1,0	0,5	1,5	4,5	0,281	27,25	0,282	124,25	0,283
X3	1,5	1,5	1,0	1,5	5,5	0,344	34,25	0,347	156	0,348
X4	0,5	0,5	0,5	1,0	2,5	0,156	14,25	0,155	64,75	0,154
Всього:					16	1	98	1	445	1

5.4 Аналіз рівня якості варіантів реалізації функцій

Визначаємо рівень якості кожного варіанту виконання основних функцій окремо.

Абсолютні значення параметрів *X2* (об’єм пам’яті для збереження даних) та *X1* (швидкодія мови програмування) відповідають технічним вимогам умов функціонування даного ПП.

Абсолютне значення параметра *X3* (час обробки даних) обрано не найгіршим (не максимальним), тобто це значення відповідає або варіанту а) 800 мс або варіанту б) 80мс.

Коефіцієнт технічного рівня для кожного варіанта реалізації ПП розраховується так (таблиця 5.6) за формулою (5.10):

$$( ) = \sum_{=1} в, \ , \ , \qquad (5.10)$$

де n – кількість параметрів;

$в$ – коефіцієнт вагомості i–го параметра;

B_i – оцінка i–го параметра в балах.

Таблиця 5.6 – Розрахунок показників рівня якості варіантів реалізації основних функцій ПП

Основні функції	Варіант реалізації функції	Абсолютне значення параметра	Бальна оцінка параметра	Коефіцієнт вагомості параметра	Коефіцієнт рівня якості
F1(X1)	А	5000	3,5	0,215	0,7525
F2(X2)	А	16	3,5	0,283	0,9905
F3(X3, X4)	А	750	2,5	0,348	0,8700
	Б	100	1	0,154	0,1540

За даними з таблиці 5.6 за формулою (5.11)

$$= ту[1] + ту[2]+\ldots + ту[\ ], \qquad (5.11)$$

де ту[ ] – коефіцієнт рівня якості і функції визначаємо рівень якості кожного з варіантів:

$К_{К1}$ = 0,774 + 0,962 + 0,835 = 2,57

$К_{К2}$ = 0,774 + 0,962 + 0,154 = 1,89

Як видно з розрахунків, кращим є перший варіант, для якого коефіцієнт технічного рівня має найбільше значення.

5.5 Економічний аналіз варіантів розробки ПП

Для визначення вартості розробки ПП спочатку проведемо розрахунок трудомісткості.

Всі варіанти включають в себе два окремих завдання:

1. Розробка проекту програмного продукту;

2. Розробка програмної оболонки;

Завдання 1 за ступенем новизни відноситься до групи А, завдання 2 – до групи Б. За складністю алгоритми, які використовуються в завданні 1 належать до групи 1; а в завданні 2 – до групи 3.

Для реалізації завдання 1 використовується довідкова інформація, а завдання 2 використовує інформацію у вигляді даних.

Проведемо розрахунок норм часу на розробку та програмування для кожного з завдань.

Проведемо розрахунок норм часу на розробку та програмування для кожного з завдань. Загальна трудомісткість обчислюється за формулою

$$То = Тр \cdot Кп \cdot Кск \cdot Км \cdot Кст \cdot Кст.м, \quad (5.12)$$

де $Тр$ – трудомісткість розробки ПП, од;

$Кп$ – поправочний коефіцієнт, од;

$Кск$ – коефіцієнт на складність вхідної інформації, од;

$Км$ – коефіцієнт рівня мови програмування, од;

$К_{СТ}$ – коефіцієнт використання стандартних модулів і прикладних програм, од;

$К_{СТ.М}$ – коефіцієнт стандартного математичного забезпечення, од.

Для першого завдання, виходячи із норм часу для завдань розрахункового характеру степеню новизни А та групи складності алгоритму 1, трудомісткість дорівнює: $Т_Р$ = 90 людино-днів. Поправочний коефіцієнт, який враховує вид нормативно-довідкової інформації для першого завдання: $К_П$ = 1.7. Поправочний коефіцієнт, який враховує складність контролю вхідної та вихідної інформації для всіх семи завдань рівний 1: $К_{СК}$ = 1. Оскільки при розробці першого завдання використовуються стандартні модулі, врахуємо це за допомогою коефіцієнта $К_{СТ}$ = 0.8. Тоді, за формулою 5.1, загальна трудомісткість програмування першого завдання дорівнює:

$$Т_1 = 90 \cdot 1.7 \cdot 0.8 = 122.4 \text{ людино-днів.}$$

Проведемо аналогічні розрахунки для подальших завдань.

Для другого завдання (використовується алгоритм третьої групи складності, степінь новизни Б), тобто $Т_Р$ =27 людино-днів, $К_П$ =0.9,$К_{СК}$ = 1,$К_{СТ}$ =0.8:

$$Т_2 = 27 \cdot 0.9 \cdot 0.8 = 19.44 \text{ людино-днів.}$$

Складаємо трудомісткість відповідних завдань для кожного з обраних варіантів реалізації програми, щоб отримати їх трудомісткість:

$$Т_I = (122.4 + 19.44 + 4.8 + 19.44) \cdot 8 = 1328{,}64 \text{ людино-годин;}$$
$$Т_{II} = (122.4 + 19.44 + 6.91 + 19.44) \cdot 8 = 1345.52 \text{ людино-годин;}$$

Найбільш високу трудомісткість має варіант II.

В розробці беруть участь два програмісти з окладом 25000 грн., один фінансовий аналітик з окладом 9000 грн. Визначимо зарплату за годину за формулою (5.13):

$$Сч = \frac{М}{\quad} \text{ грн.,} \qquad (5.13)$$

де М – місячний оклад працівників, грн;

– кількість робочих днів тиждень, од;

– кількість робочих годин в день, год.

$$Сч = \frac{25000 + 25000 + 9000}{3 \cdot 21 \cdot 8} = 117{,}06 \text{ грн.}$$

Тоді, розрахуємо заробітну плату за формулою (5.14)

$$Сзп = Сч \cdot Т \cdot Кд, \qquad (5.14)$$

де Сч– величина погодинної оплати праці програміста;
Т – трудомісткість відповідного завдання;

Кд – норматив, який враховує додаткову заробітну плату.

Зарплата розробників за варіантами становить:

I. Сзп = 117,06· 1328,64 · 1,2 = 186638,84 грн.
II. Сзп = 117,06 · 1345,52 · 1,2 = 189007,89 грн.

Відрахування на єдиний соціальний внесок становить 22%:

I. Свід = Сзп · 0.22= 186638,84 · 0,22 = 41060,54 грн.
II. Свід = Сзп · 0.22 = 189007,89 · 0,22 = 41581,74 грн.

Тепер визначимо витрати на оплату однієї машино-години. (См)

Так як одна ЕОМ обслуговує одного програміста з окладом 25000 грн., з коефіцієнтом зайнятості 0,2 то для однієї машини отримаємо:

$$С_г = 12 \cdot М \cdot К_з = 12 \cdot 25000 \cdot 0,2 = 60000 \text{ грн.}$$

З урахуванням додаткової заробітної плати:

$$С_{зп} = С_г \cdot (1 + К_з) = 60000 \cdot (1 + 0.2) = 72000 \text{ грн.}$$

Відрахування на єдиний соціальний внесок:

$$С_{від} = С_{зп} \cdot 0.3677 = 72000 \cdot 0,22 = 15840 \text{ грн.}$$

Амортизаційні відрахування розраховуємо при амортизації 25% та вартості ЕОМ – 8000 грн за формулою (5.15).

$$С_А = К_{тм} \cdot К_А \cdot Ц_{пр} = 1.15 \cdot 0.25 \cdot 8000 = 2300 \text{ грн.}, \quad (5.15)$$

де $К_{тм}$– коефіцієнт, який враховує витрати на транспортування та монтаж приладу у користувача;

$К_А$– річна норма амортизації;

$Ц_{пр}$– договірна ціна приладу.

Витрати на ремонт та профілактику розраховуємо за формулою (5.16):

$$С_Р = К_{тм} \cdot Ц_{пр} \cdot К_Р = 1.15 \cdot 8000 \cdot 0.05 = 460 \text{ грн.}, \quad (5.16)$$

де $К_Р$– відсоток витрат на поточні ремонти.

Ефективний годинний фонд часу ПК за рік розраховуємо за формулою:

$$Т_{ЕФ} = (Д_К – Д_В – Д_С – Д_Р) \cdot t_З \cdot К_В = (365 – 104 – 8 – 16) \cdot 8 \cdot 0.9 = 1706.4 \text{ годин,}$$

де $Д_К$ – календарна кількість днів у році;

$Д_В$, $Д_С$ – відповідно кількість вихідних та святкових днів;

$Д_Р$ – кількість днів планових ремонтів

устаткування; t –кількість робочих годин в день;

$К_В$– коефіцієнт використання приладу у часі протягом зміни.

Витрати на оплату електроенергії розраховуємо за формулою (5.17):

$$С_{ЕЛ} = Т_{ЕФ} \cdot N_С \cdot Ц_{ЕН} = 1706,4 \cdot 0,156 \cdot 1,93819 = 515,94 \text{ грн.}, \quad (5.17)$$

де $N_С$ – середньо-споживча потужність

приладу; $К_З$– коефіцієнтом зайнятості приладу;

$Ц_{ЕН}$ – тариф за 1 КВт-годин електроенергії.

Накладні витрати розраховуємо за формулою:

$$С_Н = Ц_{ПР} \cdot 0.67 = 8000 \cdot 0,67 = 5360 \text{ грн.}$$

Тоді, річні експлуатаційні витрати будуть за формулою (5.18):

$$С_{ЕКС} = С_{ЗП} + С_{ВІД} + С_А + С_Р + С_{ЕЛ} + С_Н \quad (5.18)$$

$$С_{ЕКС} = 72000 + 15840 + 2300 + 460 + 515,94 + 5360 = 96475,94 \text{ грн.}$$

Собівартість однієї машино-години ЕОМ дорівнюватиме:

$$С_{М\text{-}Г} = С_{ЕКС} / Т_{ЕФ} = 96475,94/1706,4 = 56,54 \text{ грн/час.}$$

Оскільки в даному випадку всі роботи, які пов‘язані з розробкою програмного продукту ведуться на ЕОМ, витрати на оплату машинного часу, в залежності від обраного варіанта реалізації, складає за формулою (5.19):

$$С_М = С_{М\text{-}Г} \cdot Т \quad (5.19)$$

I. $С_М = 56,54 \cdot 1328,64 = 75121,31$ грн.;

II. $С_М = 56,54 \cdot 1345,52 = 76075,70$ грн.;

Накладні витрати складають 67% від заробітної плати:

$С_Н = С_{ЗП} \cdot 0,67$

I. $С_Н = 186638,84 \cdot 0,67 = 125048,02$ грн.;

II. $С_Н = 189007,89 \cdot 0,67 = 125048,02$ грн.;

Отже, вартість розробки ПП за варіантами становить за формулою (5.20):

$$С_{ПП} = С_{ЗП} + С_{ВІД} + С_М + С_Н \qquad (5.20)$$

I. $С_{ПП} = 186638,84 + 41060,54 + 75121,31 + 125048,02 = 427868,71$ грн.;

II. $С_{ПП} = 189007,89 + 41581,74 + 76075,70 + 125048,02 = 431713,35$ грн.;

5.6 Вибір кращого варіанта ПП техніко-економічного рівня

Розрахуємо коефіцієнт техніко-економічного рівня за формулою (5.21):

$$К_{ТЕРj} = К_{Кj} / С_{Фj}, \qquad (5.21)$$

$К_{ТЕР1} = 2,613 / 427868,71 = 0,61 \cdot 10^{-5}$;

$К_{ТЕР2} = 1,897 / 431713,35 = 0,44 \cdot 10^{-5}$;

Як бачимо, найбільш ефективним є перший варіант реалізації програми з коефіцієнтом техніко-економічного рівня $К_{ТЕР1} = 0,61 \cdot 10^{-5}$.

5.7 Висновки до розділу 5

В даному розділі проведено повний функціонально-вартісний аналіз ПП, який було розроблено в рамках дипломного проекту. Процес аналізу можна умовно розділити на дві частини.

В першій з них проведено дослідження ПП з технічної точки зору: було визначено основні функції ПП та сформовано множину варіантів їх реалізації; на основі обчислених значень параметрів, а також експертних оцінок їх важливості було обчислено коефіцієнт технічного рівня, який і дав змогу визначити оптимальну з технічної точки зору альтернативу реалізації функцій ПП.

Другу частину ФВА присвячено вибору із альтернативних варіантів реалізації найбільш економічно обґрунтованого. Порівняння запропонованих варіантів реалізації в рамках даної частини виконувалось за коефіцієнтом ефективності, для обчислення якого були обчислені такі допоміжні параметри, як трудомісткість, витрати на заробітну плату, накладні витрати.

Після виконання функціонально-вартісного аналізу програмного комплексу що розроблюється, можна зробити висновок, що з альтернатив, що залишились після першого відбору двох варіантів виконання програмного комплексу оптимальним є перший варіант реалізації програмного продукту. У нього виявився найкращий показник техніко-економічного рівня якості $К_{ТЕР} = 0{,}12 \cdot 10^{-4}$.

Цей варіант реалізації програмного продукту має такі параметри:

мова програмування – Python

введення даних вручну (з файлу з певним форматом даних);

текстовий інтерфейс користувача.

Даний варіант виконання програмного комплексу дає користувачу зручний інтерфейс, непоганий функціонал і швидкодію.

ВИСНОВКИ

Результатом даної дипломної розробки є реалізація алгоритму пошуку оптимального набору маршрутів громадського транспорту методами колективного інтелекту та порівняння роботи цього алгоритму з іншими алгоритмами на тестовому прикладі. Продуктом даної розробки є програмний код алгоритму мовою Python.

Було проведено аналіз існуючих критеріїв оптимізації і алгоритмів вирішення даної задачі. В результаті аналізу були обрані критерії оптимізації та алгоритм вирішення задачі.

Було проведено аналіз можливих варіантів реалізації програмного продукту. В результаті найбільш економічно-вигідною виявилась розробка програми на Python з текстовим інтерфейсом.

В ході дипломної роботи були розглянуті існуючі методи вирішення задачі пошуку оптимального набору маршрутів громадського транспорту.

У результаті виконання дипломної роботи були отримані такі результати:

- Обрано критерії оптимізації.
- Логічно обгрунтовано вибір алгоритму для вирішення даної задачі.
- Було розроблено програмну реалізацію обраного алгоритму.
- Було проведено порівняння результатів роботи алгоритму на тестовому прикладі і знайдено, що результати є досить непоганими і співставними з результатами роботи інших сучасних алгоритмів.

В результаті виконання дипломної роботи було розроблено програмну реалізацію алгоритму колонії мурах для вирішення задачі пошуку оптимального набору маршрутів громадського транспорту.

Подальший розвиток можливий у таких напрямах:

- Покращення алгоритму колонії мурах шляхом більш інтелектуального підбору його параметрів
- Зміна алгоритму оновлення феромону
- Зменшення граничного порогу коефіцієнту нелінійності маршруту (і обмеження коефіцієнта нелінійності окремих частин маршруту)
- Введення «штучних» пасажирів між парами початок-кінець між якими ніхто не подорожує напряму
- Визначення цільової функції на основі кількісного визначення пріоритетів пасажира (безпересадковість або швидкість)
- Заміна алгоритму колонії мурах іншими алгоритмами колективного інтелекту
- Введення додаткових обмежень при виборі маршрутів (довжина, рівномірність завантаження тощо), в т.ч. з роботи [6]
- Паралелізація алгоритму
- Тестування алгоритму на інших прикладах з [12] або реальних даних (наприклад, Київ)

Основна галузь застосування – транспортне планування. Використовуючи розробку можливо поліпшити якість надання (зручність маршрутів і час у дорозі) транспортних послуг у місті і оптимізувати використання рухомого складу та пального для компаній-перевізників.

ПЕРЕЛІК ПОСИЛАНЬ

1. C. L. Mumford Research on the Urban Transit Routing Problem (Bus Routing). - Режим доступу: https://users.cs.cf.ac.uk/C.L.Mumford/Research%20Topics/UTRP/Outline.html. - Дата доступу: 10.05.2017.

2. P. Chakroborty Genetic Algorithms for Optimal Urban Transit Network Design / P. Chakroborty, T. Dwivedi // Engineering Optimization. – 2002 - №34 (1) - С. 83–100.

3. V. Guihairea Optimal Route Network Design for Transit Systems Using Genetic Algorithms / V. Guihairea, Jin-Kao Hao // Transportation Research Part A: Policy and Practice. – 2008. - №42. – С. 1251–1273

4. O.J. Ibarra-Rojas Planning, operation, and control of bus transport systems: A literature review / O.J. Ibarra-Rojas, F. Delgado, R. Giesen and J.C. Munoz // Transportation Research Part B: Methodological. – 2015. - №77. - С. 38–75

5. R.Z. Farahani A review of urban transportation network design problems / R.Z. Farahania, E. Miandoabchib, W.Y. Szetoc, H. Rashidid // European Journal of Operational Research. – 2013. - №229 (2). - С. 281–302

6. Z. Yang A Parallel Ant Colony Algorithm for Bus Network Optimization / Z. Yang, Bin Yu, C. Cheng // Computer-Aided Civil and Infrastructure Engineering. – 2007. - №22. – С. 44-55

7. K. Kepaptsoglou Transit Route Network Design Problem: Review / K. Kepaptsoglou, M. Karlaftis // Journal of Transportation Engineering. – 2009. - №135(8) – С. 491-505

8. H. Gunby A Combined Swarm System for the Urban Transit Routing Problem / H. Gunby, S. Gustavsen – 2005 – 111 с.

9. M. P. John An Improved Multi-objective Algorithm for the Urban Transit Routing Problem / M. P. John, C. L. Mumford, and R. Lewis // Evolutionary Computation in Combinatorial Optimisation. EvoCOP 2014. Lecture Notes in Computer Science, vol 8600. - Springer, Berlin, Heidelberg. – С. 49-60

10. L. Fan A Simple Multi-Objective Optimization Algorithm for the Urban Transit Routing Problem / L. Fan, C. L. Mumford, D. Evans // Evolutionary Computation, 2009. CEC '09. IEEE Congress on, 18-21 May 2009. - Trondheim, Norway – С. 100-120

11. P. N. Kechagiopoulos Solving the Urban Transit Routing Problem using a particle swarm optimization based algorithm / P. N. Kechagiopoulos, G. N. Beligiannis / Applied Soft Computing. – 2014. - №22 – С. 654-676

12. C. L. Mumford New Heuristic and Evolutionary Operators for the Multi-Objective Urban Transit Routing Problem / C.L. Mumford // Evolutionary Computation (CEC), 2013 IEEE Congress on, 20-23 June 2013. - Cancun, Mexico. – С. 151-178

13. L. Fan A metaheuristic approach to the urban transit routing problem / L. Fan, C. L. Mumford // Jounal of Heuristics. – 2010. - №16. – С. 353-372

14. Рак О.О. Побудова мережі маршрутів громадського транспорту як задача багатокритеріальної оптимізації / Рак О.О. // Системний аналіз та інформаційні технології: матеріали 19-ї Міжнародної науково-технічної конференції SAIT 2017, Київ, 22 – 25 травня 2017 р. – К.: ННК “ІПСА” НТУУ “КПІ ім. Ігоря Сікорського”, 2017. – С. 110

15. A. B. Nunez Sustainable urban transport for Kyiv: towards a sustainable and competitive city built upon the legacy system and innovations - Режим доступу: http://documents.worldbank.org/curated/en/640531472066198963/Sustainable-urban-transport-for-Kyiv-towards-a-sustainable-and-competitive-city-built-upon-the-legacy-system-and-innovations. - Дата доступу: 10.05.2017.

16. C.E. Mandl. Applied Network Optimization / C.E. Mandl. – Academic Press, London, 1979. – 185 с.

17. M.H. Baaj An AI-based approach for transit route system planning and design / M.H. Baaj, H.S. Mahmassani // Journal of Advanced Transportation. - №25 (2). – 1991. – С. 187–209

18. Kidwai F. A. Optimal design of bus transit network: a genetic algorithm based approach / Kidwai F. A // Department of Civil Engineering. Kanpur, India, IIT Ph. D. – 1998.

19. Chakroborty P. Optimal route network design for transit systems using genetic algorithms / Chakroborty P., Wivedi T. // Engineering optimization. – 2002. – №34 (1) – С. 83-100.

20. Zhang J. The multi-objective optimization algorithm to a simple model of urban transit routing problem / Zhang J., Lu H., Fan L. // Natural Computation (ICNC), 2010 Sixth International Conference on. – IEEE, 2010. – №6. – С. 2812-2815.

21. Chew J. S. C. A genetic algorithm for urban transit routing problem / Chew J. S. C., Lee L. S. //International Journal of Modern Physics: Conference Series. – World Scientific Publishing Company, 2012. – № 9. – С. 411- 421.

22. Мартынова Ю. А. Анализ опыта проектирования рациональных маршрутных сетей городского пассажирского транспорта / Мартынова Ю. А. // Интернет-журнал Науковедение. - 2014. - №2 (21). - С.125

23. Мартынова Ю. А. Формализация задачи организации маршрутных сетей городского пассажирского транспорта / Мартынова Ю. А., Мартынов Я. А. // Интернет-журнал Науковедение. – 2014. – №. 6 (25).

24. Dorigo M. Optimization, learning and natural algorithms / Dorigo M. // Ph. D. Thesis, Politecnico di Milano, Italy. – 1992.

25. Dorigo M. Ant system: optimization by a colony of cooperating agents / Dorigo M., Maniezzo V., Colorni A. // IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics). – 1996. – №26 (1) – С. 29-41.

26. Stützle T. The max-min ant system and local search for combinatorial optimization problems / Stützle T., Hoos H. // Meta-heuristics, Springer US – 1999. – С. 313-329.

27. Badr A. A proof of convergence for ant algorithms / Badr A., Fahmy A. A // Information Sciences. – 2004. – №160 (1) – С. 267-279.

ДОДАТОК А. РЕАЛІЗАЦІЯ АЛГОРИТМУ МОВОЮ PYTHON

Реалізація алгоритму ACA мовою Python:

```
from random import choices

# TODO: add Ant-Weight coeffs, define INITIAL_PHEROMONES
const alpha = 1.0
beta = 2.0
Q = 10.0

ro = 0.3

index = 0

def get_adj_nodes(node_no):
    return [key[1] for key in graph.keys() if key[0] == node_no]

file = open('log.csv', 'w')

class Ant:
    def __init__(self, start, finish, graph, global_graph_section_flows,
local_graph_section_flows, global_pheromone_matrix, local_pheromone_matrix):
        self.start = start
        self.finish = finish

        self.graph = graph
        self.global_pheromone_matrix = global_pheromone_matrix
        self.local_pheromone_matrix = local_pheromone_matrix
        self.global_graph_section_flows = global_graph_section_flows
        self.local_graph_section_flows = local_graph_section_flows

        self.is_lost = False
        self.i = start
        self.tabu_list = [self.start]

        self.L = 0.0

    def __str__(self):
        return str(self.start) + ", " + str(self.finish)

    def set_route_i(self, route_i):
        self.route_i = route_i

    def calculate_step(self):
        if self.is_lost is True:
            return

        if self.i == self.finish:
            return

        p_list = []
        S = 0
        adj_nodes = [j for j in get_adj_nodes(self.i) if j not
in self.tabu_list]

        for j in adj_nodes:
```

```
            nu_ij = self.global_graph_section_flows[(self.i, j)]
/ self.graph[(self.i, j)]
            tau_ij = self.global_pheromone_matrix[(self.i, j)]
            global alpha
            global beta

            global file
            global index
            file.write(str(index) + ';' + str(tau_ij ** alpha) + ';'
+ str(nu_ij ** beta) + '\n')
            index += 1

            p = tau_ij ** alpha + nu_ij ** beta
            p_list.append(p)
            S += p

        p_list = [p / S for p in p_list]

        try:
            j = choices(adj_nodes, p_list)[0]
            self.tabu_list.append(j)
        except:
            self.is_lost = True

        if self.is_lost is True:
            return

        self.L += graph[(self.i, j)]
        self.i = j


### START reading OD

matrix od_sum = 0

f = open('Mandl/MandlDemand.txt', 'r')
od_matrix = []
for line in f.readlines():
    if line == '\n':
        continue
    row = [int(item) // 10 for item in line.replace('\n', '').split(' ')
if item != '']
    od_matrix.append(row)
    od_sum += sum(row)

### FINISH reading OD matrix

### START reading travel times

f = open('Mandl/MandlTravelTimes.txt', 'r')
network = []
for line in f.readlines():
    if line == '\n':
        continue
    pre_row = [item for item in line.replace('\n', '').split(' ') if item !=
'']
    row = [int(item) if item != 'Inf' else None for item in
    pre_row] network.append(row)

# print(network)

### FINISH travel times
```

```
### START creating graph with travel times

graph = {}
for i in range(0, len(network)):
    for j in range(0, len(network)):
        if network[i][j] is None:
            continue
        if i == j:
            continue
        graph[(i, j)] = network[i][j]

print(graph)

### FINISH creating graph with travel times

while True:
    ### START initialize pheromone

    matrices INITIAL_PHEROMONES = 10.0

    global_pheromone_matrix = {}
    for key in graph.keys():
        global_pheromone_matrix[key] = INITIAL_PHEROMONES

    local_pheromone_matrices = {}
    for key in graph.keys():
        local_pheromone_matrices[key] = {}

    ### FINISH initialize pheromone matrices

    ### START initialize section flows counting

    global_graph_section_flows = {}
    for key in graph.keys():
        global_graph_section_flows[key] = 0

    local_graph_section_flows = {}
    for key in graph.keys():
        local_graph_section_flows[key] = {}

    ### FINISH initialize section flows counting

    iter_no = 100
    all_ants_no = 0
    colonies = []

    for row in range(0, len(od_matrix)):
        for col in range(row + 1, len(od_matrix)):
            if row != col and od_matrix[row][col] != 0 and od_matrix[col][row]:
                colonies.append((row, col))

    colony_no = len(colonies)

    f = True

    routes = []

    for iter_i in range(iter_no):
        successful_ants = []

        #print("iter_i: " + str(iter_i) + ' / ' + str(iter_no))

        for colony_i in range(colony_no):
```

```
            ants_no = od_matrix[colonies[colony_i][0]][colonies[colony_i][1]]
+ od_matrix[colonies[colony_i][1]][colonies[colony_i][0]]
            ants = [Ant(colonies[colony_i][0], colonies[colony_i][1], graph,
global_graph_section_flows, local_graph_section_flows, global_pheromone_matrix,
local_pheromone_matrices) for ant_i in range(ants_no)]

            for k in range(0, 100):
                for ant in ants:
                    ant.calculate_step()

                    if ant.i == ant.finish:
                        successful_ants.append(ant)
                        ants.remove(ant)

        for ant in successful_ants:

            route_i = None
            if ant.tabu_list in routes:
                route_i = routes.index(ant.tabu_list[:])
            else:
                routes.append(ant.tabu_list[:])
                route_i = len(routes) - 1

            for i in range(0, len(ant.tabu_list) - 1):
                global_graph_section_flows[(ant.tabu_list[i], ant.tabu_list[i +
1])] += 1
                try:
                    local_graph_section_flows[(ant.tabu_list[i],
ant.tabu_list[i + 1])][route_i] += 1
                except:
                    local_graph_section_flows[(ant.tabu_list[i],
ant.tabu_list[i + 1])][route_i] = 1

            for i in range(0, len(ant.tabu_list) - 1):
                f_ij = global_graph_section_flows[(ant.tabu_list[i],
ant.tabu_list[i + 1])]
                fk_ij = local_graph_section_flows[(ant.tabu_list[i],
ant.tabu_list[i + 1])][route_i]
                try:
                    local_pheromone_matrices[(ant.tabu_list[i],
ant.tabu_list[i + 1])][route_i] += \
                        Q / f_ij * (f_ij - fk_ij) / ((len(ant.tabu_list) *
f_ij))
                except:
                    local_pheromone_matrices[(ant.tabu_list[i], ant.tabu_list[i
+ 1])][route_i] = \
                        Q / f_ij * (f_ij - fk_ij) / ((len(ant.tabu_list) *
f_ij))

        for key in global_pheromone_matrix.keys():
            global_pheromone_matrix[key] *= 1.0 -
            ro global_pheromone_matrix[key] +=
sum([local_pheromone_matrices[key][route_i] for route_i in range(len(routes))
if route_i in local_pheromone_matrices[key].keys()])

    dtd_list = {}
    l_list = {}
    dt_list = {}
    for j in range(len(routes)):
        l = sum([graph[(routes[j][i], routes[j][i + 1])] for i
in range(len(routes[j]) - 1)])
        dt = 0.0
```

```
        for colony in colonies:
            if colony[0] in routes[j] and colony[1] in routes[j]:
                dt += od_matrix[colony[0]][colony[1]] +
od_matrix[colony[1]][colony[0]]

        min_l = 999999999
        min_l_i = 0

        f = True

        for k in range(len(routes)):
            if routes[k][0] == routes[j][0] and routes[k][len(routes[k]) -
1] == routes[j][len(routes[j]) - 1]:
                new_l = sum([graph[(routes[k][i], routes[k][i + 1])] for i
in range(len(routes[k]) - 1)])
                if new_l < min_l:
                    min_l = new_l
                    min_l_i = k

            dtd_list[j] = dt / l
            dt_list[j] = dt
            l_list[j] = l

    max_dtd = 0
    best_route_i = 0
    for key in dtd_list.keys():
        if dtd_list[key] > max_dtd and len(routes[key]) >= 4:
            max_dtd = dtd_list[key]
            best_route_i = key

    print(str(routes[best_route_i]) + ' - ' + str(dt_list[best_route_i]) + '
/ ' + str(od_sum) + ' - ' + str(dtd_list[best_route_i]))

    for colony in colonies:
        if colony[0] in routes[best_route_i] and colony[1]
in routes[best_route_i]:
            od_matrix[colony[0]][colony[1]] = 0
            od_matrix[colony[1]][colony[0]] = 0

    od_sum = 0
    for row in od_matrix:
        od_sum += sum(row)

    for row in range(len(od_matrix)):
        for col in range(len(od_matrix)):
            if od_matrix[row][col] > 1:
                f = False

    if f is True:
        break
```

Реалізація аналізу набору маршрутів мовою Python:

```
import networkx as nx

### START reading OD matrix

f = open('Mandl/MandlDemand.txt', 'r')
od_matrix = []
for line in f.readlines():
    if line == '\n':
        continue
    row = [int(item) for item in line.replace('\n', '').split(' ') if item !=
''']
    od_matrix.append(row)

### FINISH reading OD matrix

### START reading travel times

f = open('Mandl/MandlTravelTimes.txt', 'r')
network = []
for line in f.readlines():
    if line == '\n':
        continue
    pre_row = [item for item in line.replace('\n', '').split(' ') if item !=
''']
    row = [int(item) if item != 'Inf' else None for item in
    pre_row] network.append(row)

# print(network)

### FINISH travel times

### START creating graph with travel times

graph = {}
for i in range(0, len(network)):
    for j in range(0, len(network)):
        if network[i][j] is None:
            continue
        if i == j:
            continue
        graph[(i, j)] = network[i][j]

print(graph)

def get_adj_nodes(graph, node_no):
    return [key[1] for key in graph.keys() if key[0] == node_no]

### FINISH creating graph with travel times

def validate_routes_set():
    global od_matrix
    global network
    global graph

    routes = [
        [3, 1, 2, 5, 7, 14, 6, 9],
        [9, 10, 12, 13],
        [0, 1, 2, 5, 7, 14, 6, 9, 10, 11, 3, 4],
        [8, 14, 6, 9],
        [0, 1, 2, 5, 7, 14, 8],
        [0, 1, 2, 5, 3, 11, 10, 12],
```

```
        [7, 5, 14, 6, 9, 13, 12]
    ]

    nodes = set()
    for route in routes:
        for node in route:
            nodes.add(node)

    print("Nodes covered: " + str(len(nodes)) + " / " + str(len(od_matrix[0]))
+ ": " + str(nodes))

    nodes = {}
    for i in range(len(routes)):
        for node in routes[i]:
            try:
                nodes[node].append(i)
            except:
                nodes[node] = [i]

    new_network = nx.DiGraph()
    new_nodes = {}

    for route in range(len(routes)):
        for i in range(len(routes[route]) - 1):
            node_1 = routes[route][i]
            node_2 = routes[route][i + 1]
            new_node_1 = str(node_1) + "-" + str(route)
            new_node_2 = str(node_2) + "-" + str(route)

            try:
                new_nodes[node_1].add(new_node_1)
            except:
                new_nodes[node_1] = set()
                new_nodes[node_1].add(new_node_1)

            try:
                new_nodes[node_2].add(new_node_2)
            except:
                new_nodes[node_2] = set()
                new_nodes[node_2].add(new_node_2)

            if route in nodes[node_2]:
                new_network.add_edge(new_node_1, new_node_2,
weight=network[node_1][node_2])
                new_network.add_edge(new_node_2, new_node_1,
weight=network[node_2][node_1])

    for node in new_nodes.keys():
        n = list(new_nodes[node])
        for i in range(len(n)):
            for j in range(i + 1, len(n)):
                new_network.add_edge(n[i], n[j], weight=5)
                new_network.add_edge(n[j], n[i], weight=5)


    def shortest_path_and_length(new_network, new_nodes, source,
target, weight='weight'):
        for node in [source, target]:
            n = list(new_nodes[node])
            for i in range(len(n)):
                new_network.add_edge(node, n[i], weight=0)
                new_network.add_edge(n[i], node, weight=0)
```

```
        results = {
            'path': nx.shortest_path(new_network, source=source,
target=target, weight=weight),
            'length': nx.shortest_path_length(new_network,
source=source, target=target, weight=weight)
        }

        for node in [source, target]:
            n = list(new_nodes[node])
            for i in range(len(n)):
                new_network.remove_edge(node, n[i])
                new_network.remove_edge(n[i], node)

        return results


    def check_transfers(path):
        transfers_no = 0
        for i in range(1, len(path) - 2):
            node_1 = path[i].split('-')[0]
            node_2 = path[i+1].split('-')[0]
            if node_1 == node_2:
                transfers_no += 1

        return transfers_no

    all_travels = 0
    all_time = 0
    transfers_dict = {
        0: 0,
        1: 0,
        2: 0,
        3: 0
    }

    for i in range(len(od_matrix) - 1):
        for j in range(i + 1, len(od_matrix[i])):
            try:
                shortest_path_and_length_data =
shortest_path_and_length(new_network, new_nodes, source=i, target=j,
weight='weight')
                time = shortest_path_and_length_data['length']
                transfers =
check_transfers(shortest_path_and_length_data['path'])

                all_travels += od_matrix[i][j] + od_matrix[j][i]
                all_time += time * (od_matrix[i][j] + od_matrix[j][i])
                transfers_dict[transfers] += od_matrix[i][j] + od_matrix[j][i]
            except:
                print("ALARM")
                transfers_dict[3] += 1

    for key in transfers_dict.keys():
        print(str(key) + " transfers: " + str(transfers_dict[key]
/ all_travels))
    print(all_time / all_travels)


validate_routes_set()
```